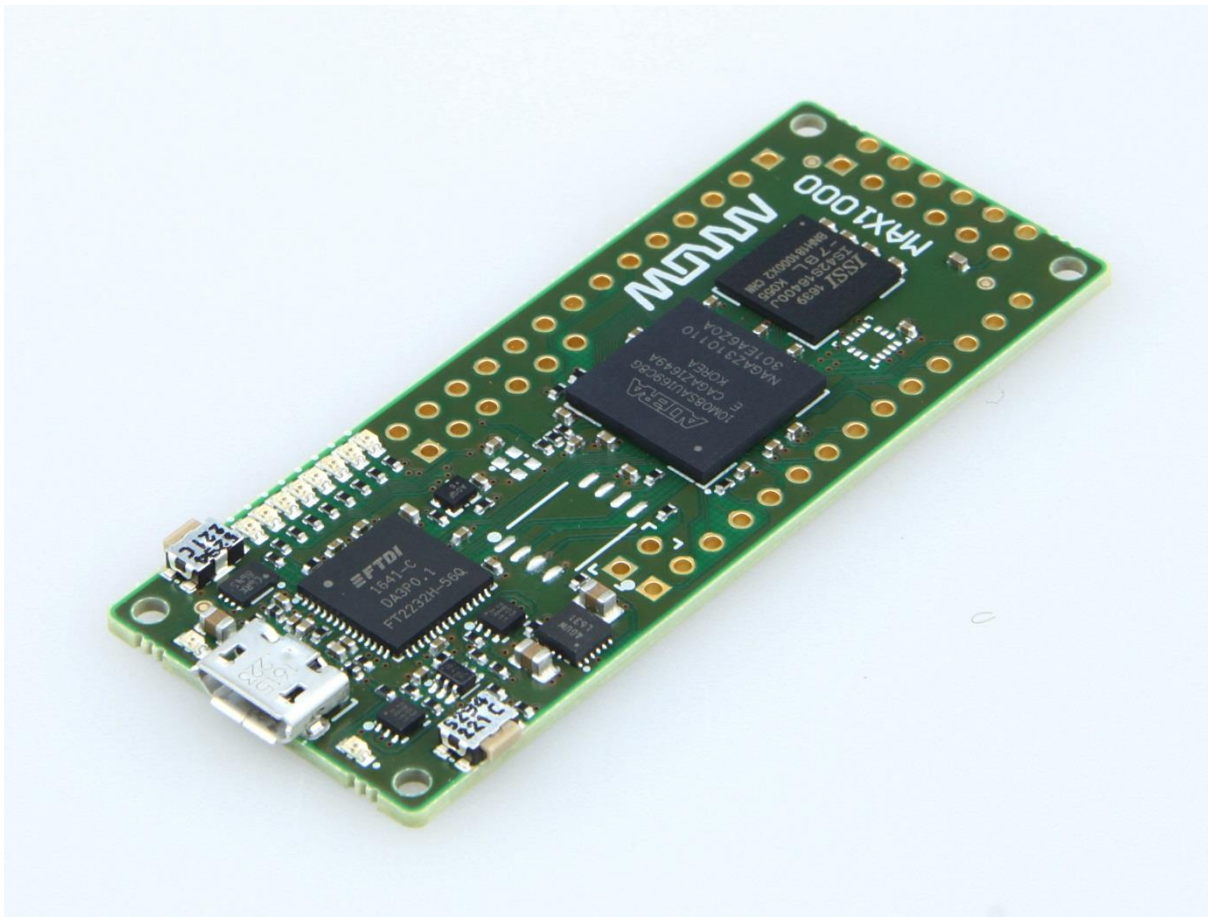


MAX1000

Workshop LABs



Chapter 1 - Lab 1. FPGA Introduction LAB

1.1 Using the Arrow MAX1000 Board.

This LAB's intention is to give you an introduction to FPGA and how to use the tools to program it.

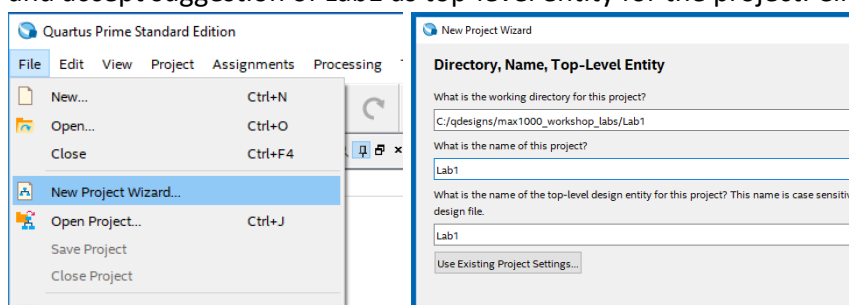
The document "MAX1000 User Guide" contains detailed information about the features of the board, and how they are connected.

Prerequisites:

- Installed Quartus Prime Lite 17.1 available at <http://dl.altera.com/?edition=lite>
- Requirements for PC for use with Max10 device family: Min 14GB free disk space. Main 2GB physical RAM.
- Extract MAX1000_workshop_labs.zip to a suitable location on your PC. (Make sure there are no "whitespaces" in the paths as some of the tools are Linux based.)
- Install the Arrow USB Blaster driver located in the extracted .zip file, or located on this link: <https://wiki.trenz-electronic.de/display/PD/MAX1000+Arrow+USB+Blaster>

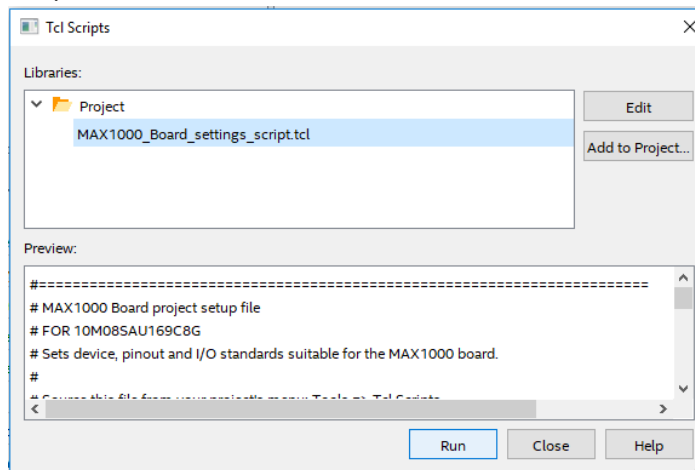
1.2 Create a Quartus Prime project

- Start Quartus Prime Lite.
- From the File menu, select "New Project Wizard". Select directory Lab1 from the extracted .zip as the working directory for your project. Type Lab1 as name of project, and accept suggestion of Lab1 as top-level entity for the project. Click Next.

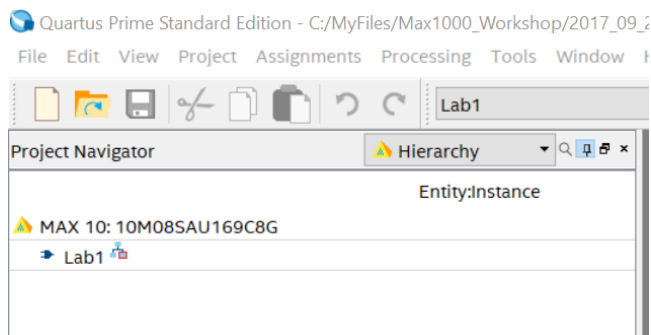


- Select Empty project and click Next
- Accept the rest of the default suggestions by clicking Finish. (We will instead source a script file to adjust these settings)
- Source the file provided in the Lab1 directory called "MAX1000_Board_settings_script.tcl" by opening the Tools menu and select "Tcl

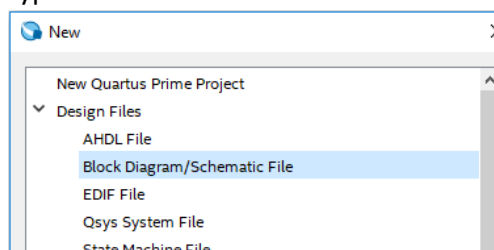
Scripts". Select our file, and click the Run button.



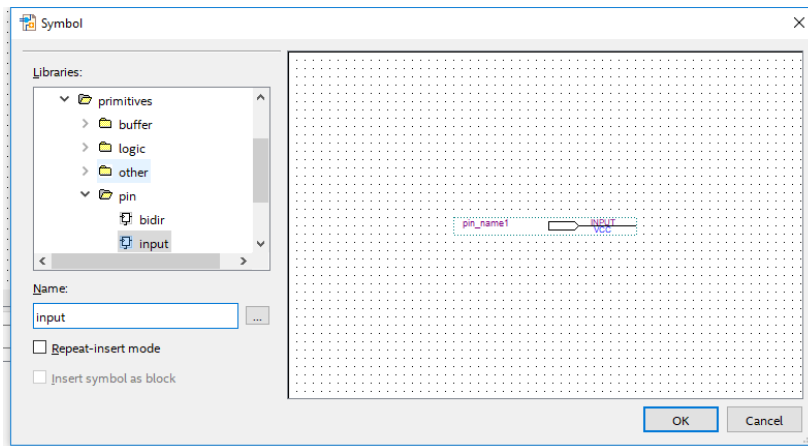
Your Project Navigator should now look like this:



- Create a new top level file: From the File menu, select "New". Choose "Block Diagram" type.



- Double click anywhere in the Schematic editor. Type "input" in the "Name" field. You will now get a symbol for an input pin suggested. Hit Enter key and place the input pin at your desired place in the schematic.

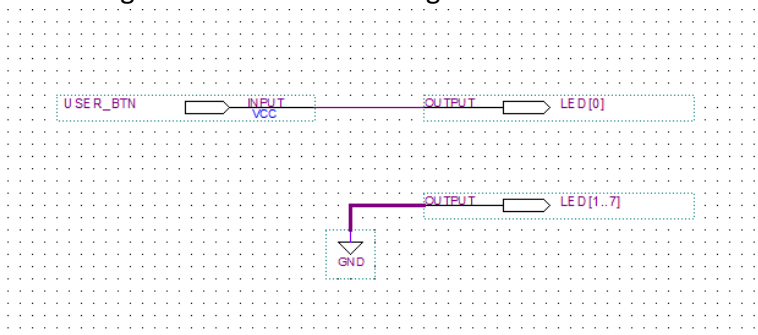


- Double click the “pin_name” text next to the symbol and change this to “USER_BTN”.
(The name is important because it has to match the names we have given the I/O assignments from the script file)



- Double click an empty place in the schematic and type “output” in the Name field. Populate your schematic with two called LED[0] and LED[1..7]
- Add a “GND” symbol to the schematic, and connect this to the LED[1..7] by selecting the “inbound wire” of the output pin symbol and dragging a wire that connects to the GND symbols wire stub.
- Save your file as Lab1.bdf (bdf=block design file)

Your design should look something like this:



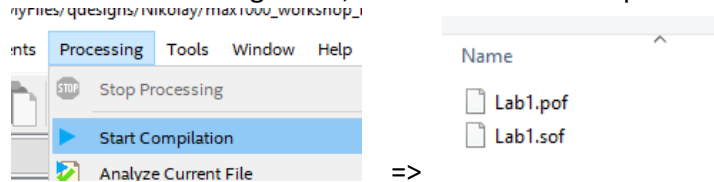
Notes:

The LEDs are lit when FPGA output drives high. The pushbutton inputs on the FPGA are pulled to Vccio with a weak resistor when not pressed, and connected to GND when pressed. The LED will therefore go dark when you push the button. (We could have inserted an inverter in our design if we wanted it to work the other way around...)

It is necessary to connect the unused LEDs in this case to GND to shut them off. Unused FPGA pin's default behaviour is to be tristated with a weak pull up resistor. The LEDs would therefore have glowed dimly.

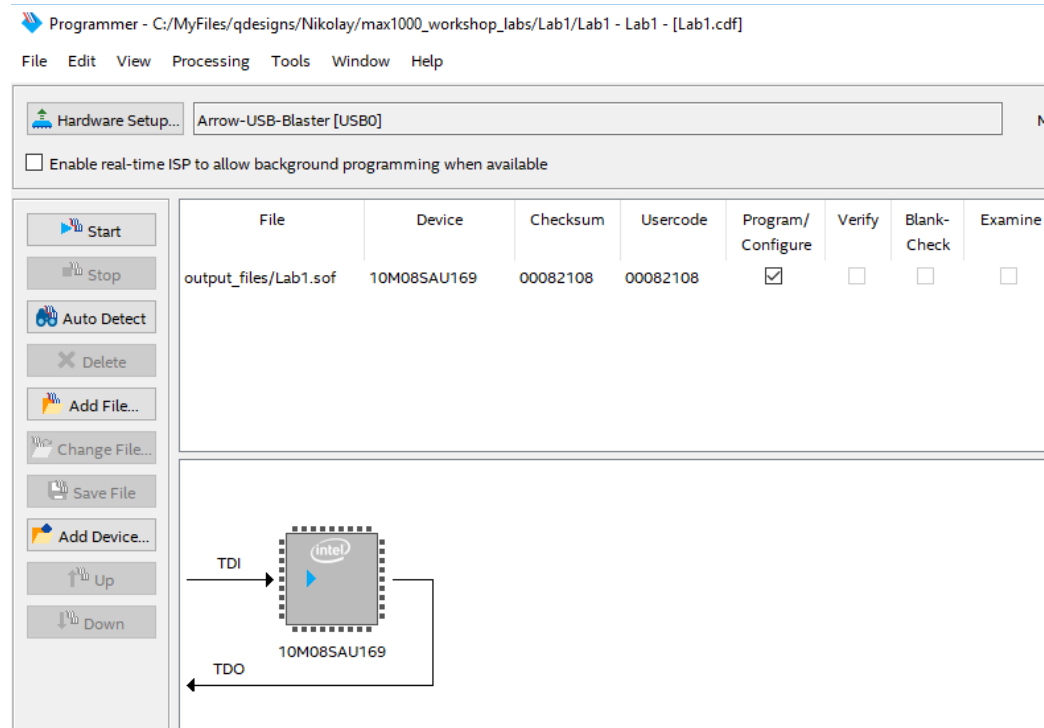
The Next step is to convert your schematic description into a “configuration file” for the FPGA. This is called “compilation”.

- From the Processing menu, select the “Start Compilation” button.



When Finished, Quartus will have produced a .sof and a .pof file. A .sof file only “configures” the FPGA’s volatile configuration bits, while a .pof file programs the on-chip configuration flash, which is automatically loaded every time the fpga is powered on or reset. Using the .sof is quick and sufficient during development phase.

- Plug in the Max1000 board.
- Open the Quartus Programmer from the menu; Tools => Programmer.
- Unless Arrow-USB-Blaster shows up automatically, you need to select this by pressing the “Hardware Setup” button.
- If your Lab1.sof has not already been added, with relation to Device 10M08SAU169, please press the “Add File” button and find it in directory “output_files”.
- Click the Start button in the Programmer window.



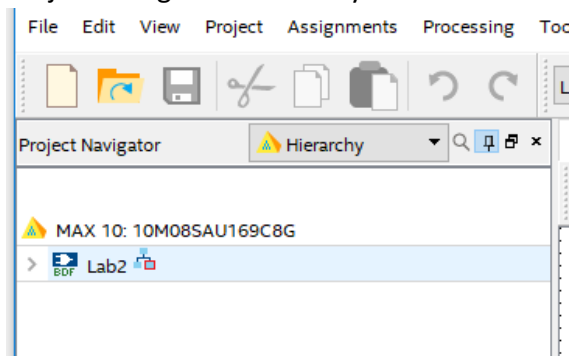
- Press the button on the evaluation kit to blink the LED0

Chapter 2 - Lab 2. Use Macro Functions

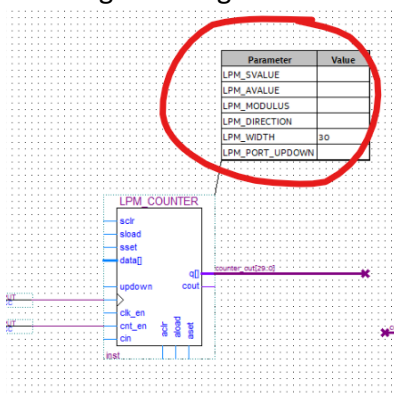
2.1 Finish the provided design

In this Lab we have provided you with an almost finished design. You will insert the missing pieces and compile. It will count binary on the 8 LEDs.

- From Quartus menu File => Open Project select the "Quartus Project File" \Lab2\Lab2.qpf
- Open the provided schematic design file by double clicking on the Lab2 entity in the Project Navigator "Hierarchy View"

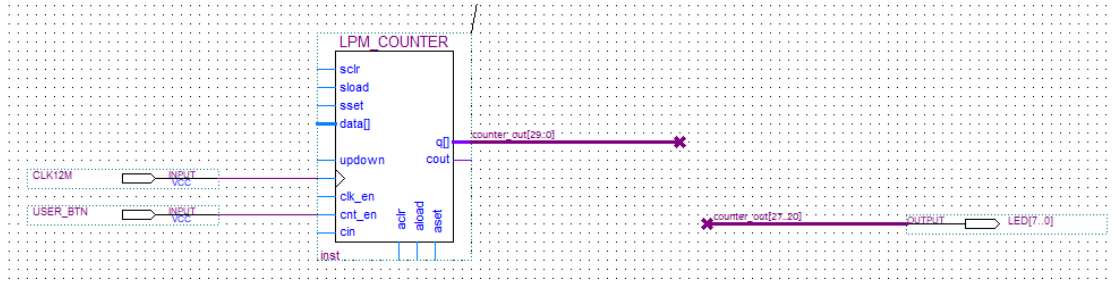


- Double click and add component LPM_COUNTER
- Double click the Parameter table on the LPM_COUNTER and change the LPM_WIDTH to 30 Unsigned Integer



- Connect the CLK12M input to the LPM_COUNTER's clock port
- Connect the USER_BTN to cnt_en port of LPM_COUNTER
- Draw a wire out(ending in "free air") from LPM_COUNTER's q[] port and name it counter_wires[29..0]. Just start typing while it is selected (blue).

- Draw a wire in from the LED[7..0] output pins and name it “counter_wires[29..22]



- Compile the design
- Program the Max1000 board

Question:

Why does it count so slow when we have a 12MHz clock?

Chapter 3 - Lab 3. Insert a soft processor

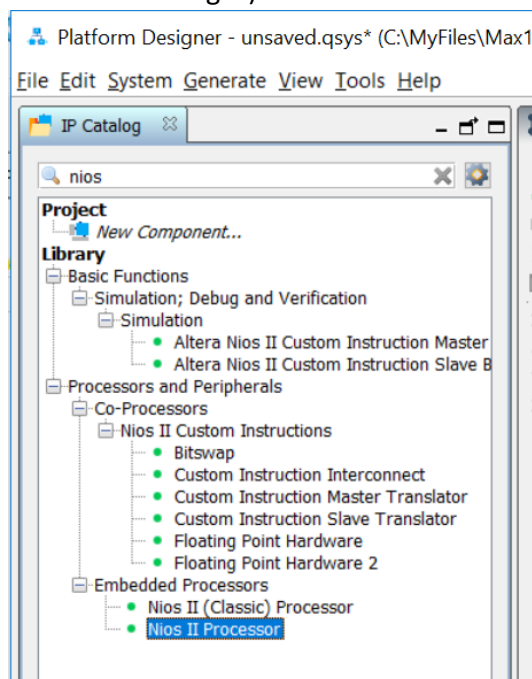
3.1 Finish the provided design

In this lab you will insert a processor in a provided framework, and write some software for it.

- Open project "Lab3.qpf"
- Open the Lab3 top level design file and examine it. Notice we have inserted a pll on the incoming clock that boost it to 24MHz. (JTAG debug requires minimum 20MHz)

3.2 Create a NiosII processor system

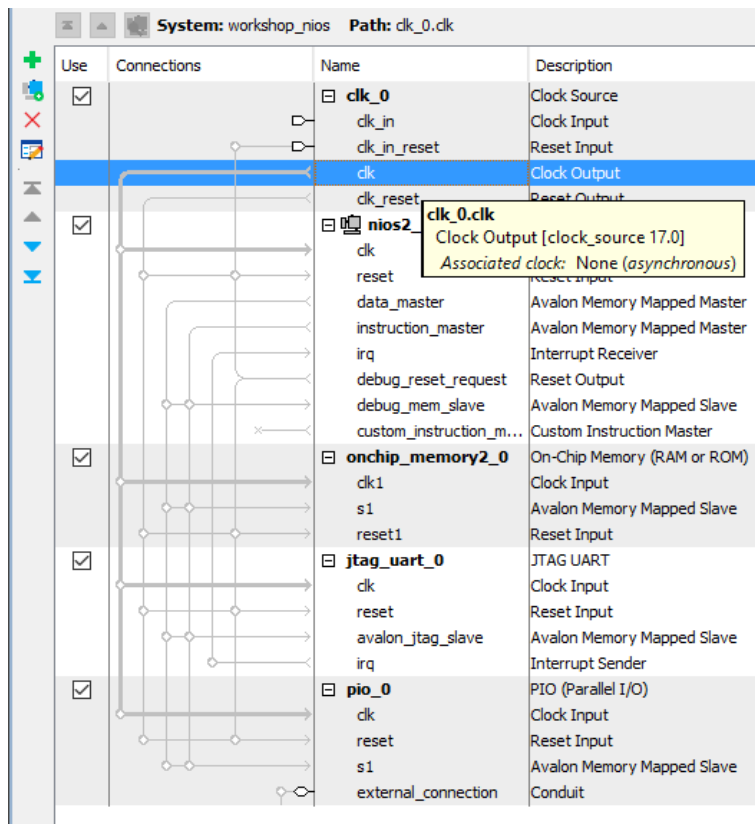
- Open "Platform Designer" from the Tools menu
- Add the following components to your Platform Designer system:
 - o Type "nios" in the search field and double click the "Nios II Processor". Select Nios II /e version. Accept the other settings as defaults and click Finish. (Despite it has red error messages)



- o Add a "System ID Peripheral" with default settings to the Qsys system.
- o Add an "On-Chip RAM" by filtering on "RAM". Change "Total Memory Size" by typing "16k". (It will convert it to 16384 bytes)
- o Add a "PIO" peripheral to the system; WIDTH=1, Output
- o Add a "JTAG UART" with default settings.

We now have a set of components but they are not yet connected. There are a lot of red error messages.

- Click the line “clk” (Clock Output) of clk_0. It highlights which other components can be connected. Connect them all.



- Do the same with “clk_reset”.
- Click the “control_slave” interface of the “System ID Peripheral” and let it be mastered by the “Nios II Processor’s” “data_master” interface.
- Click the “s1” interface of the “On-Chip Memory” and connect it to BOTH data_master and instruction_master interfaces of the processor.
- Pio_0’s s1 interface connects to NiosII data_master
- Jtag_uart_0 avalon_jtag_slave connects to NiosII data_master
- Jtag_uart_0 irq connects to NiosII irq
- Save the Platform Designer system as “workshop_nios” or similar

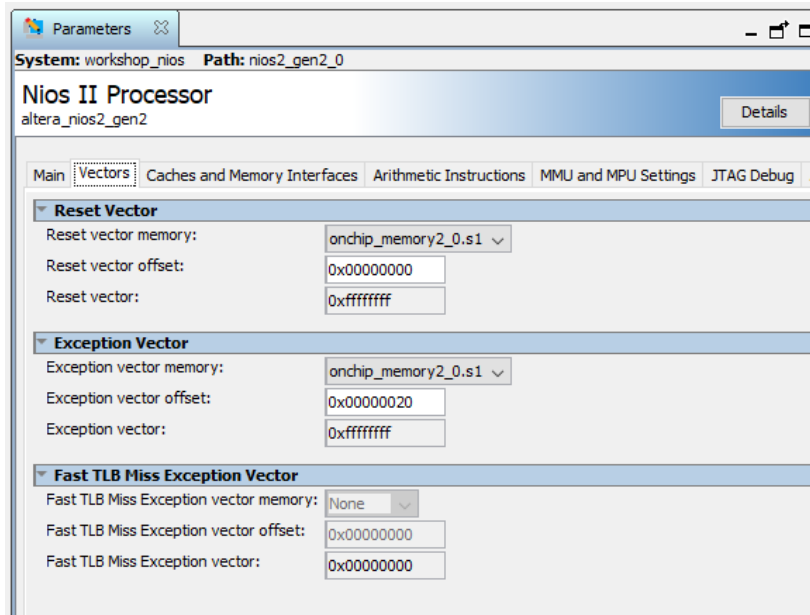
Since we added peripherals without caring about memory address space, we have a lot of overlap and need to fix it:

- On the “System” menu click “Assign Base Addresses”.

Only two errors left. Where are the processors reset and exception vectors pointing?

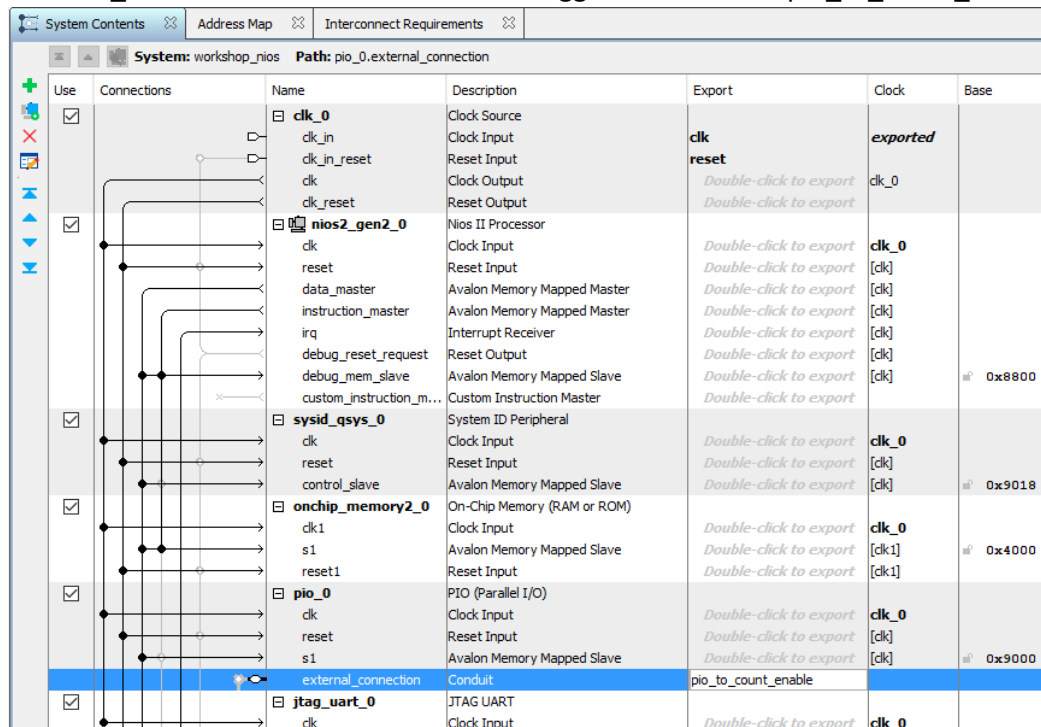
- Double click the nios2_gen2_0 line and watch a Parameters window appear on the right.
- Click the “Vectors” tab and assign “Reset vector memory” to “onchip_memory2_0.s1”

- Do the same for “Exception vector memory”

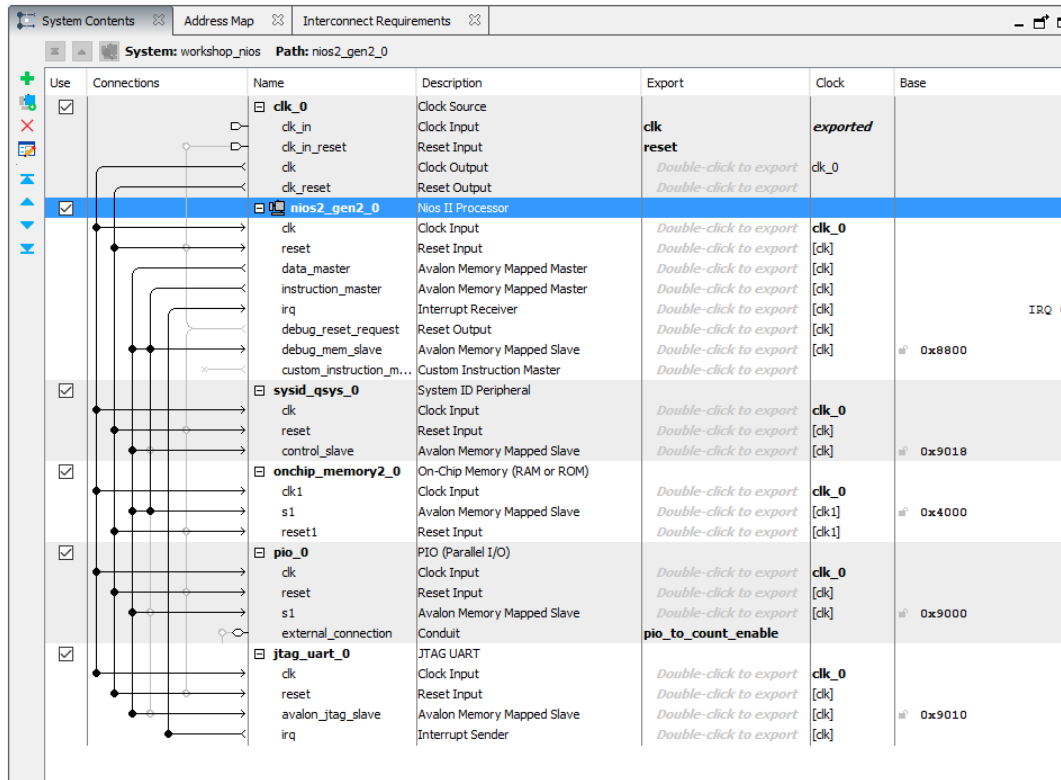


The last warning: pio_0's “external_connection” is not yet exported...

- In the System Contents “Export” column, double click on the pio_0's external_connection line and rename the suggested name to “pio_to_count_enable”



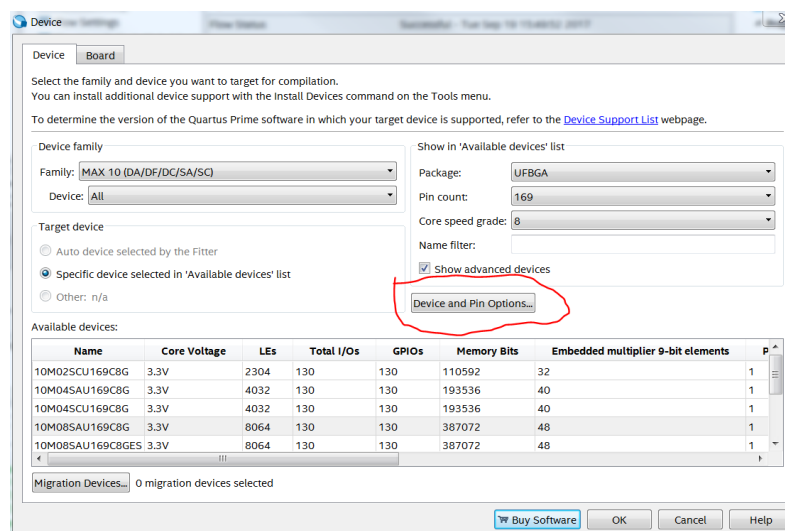
Now it should be all green and look something like this:



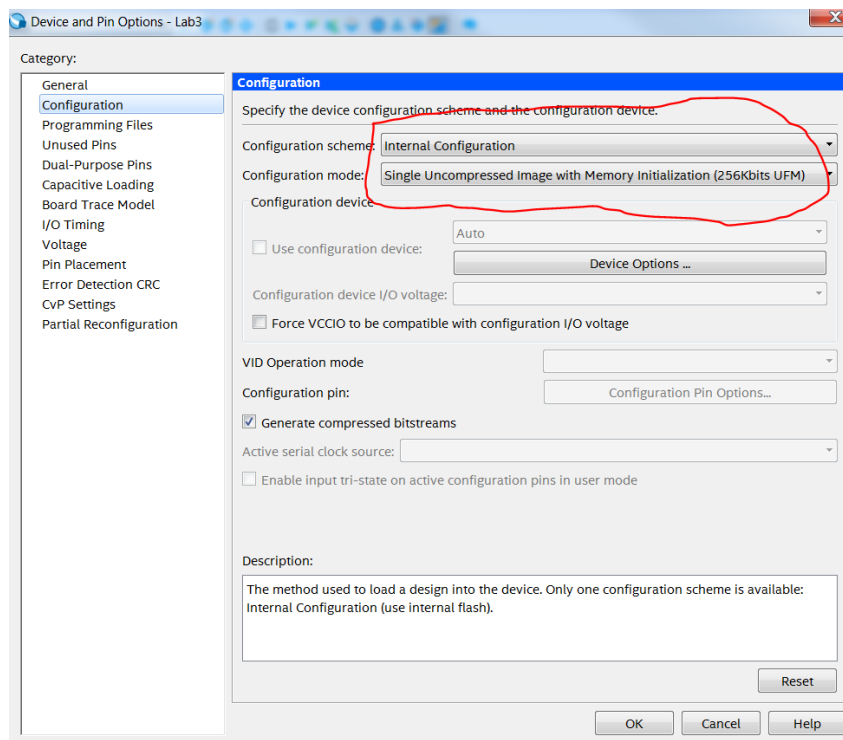
- Click "Generate HDL" and accept the defaults by clicking "Generate".

When generate completes it is time to switch back to Quartus and insert the processor system into our design.

- Add the newly created processor system files to the project by clicking menu Project => Add/Remove Files In Project. Browse and select the file
\\workshop_nios\\synthesis\\workshop_nios.qip
- Double click in the schematic editor, and click the browse button next to the "Name" textbox. Select the workshop_nios that should appear. Hopefully the ports on the symbol should fit directly onto the open stubs in the schematic.
- Check that reset on the symbol is connected to VCC.
- Tell Quartus that NIOS will boot from internal RAM by selecting \\Device\\Device and Pin Options\\ from the assignments menu as shown on picture below.



- In the “Device and Pin Options” select Configuration left and then the settings as shown on picture below.



- Compile the project and program the Max1000 board.

3.3 Creating a software project for the processor.

We have now created a custom processor. How does any compiler know what we have created?

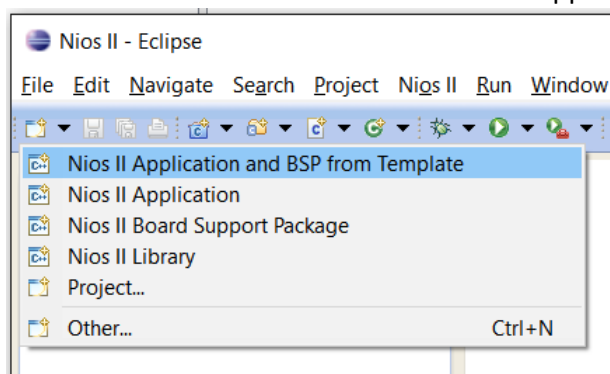
The NiosII EDS is based on the open source tools Eclipse and gcc/gdb but with some additional functionality to handle the customizability.

There are multiple templates that cover everything from bare-metal projects to applications on top of RTOS and Linux variants.

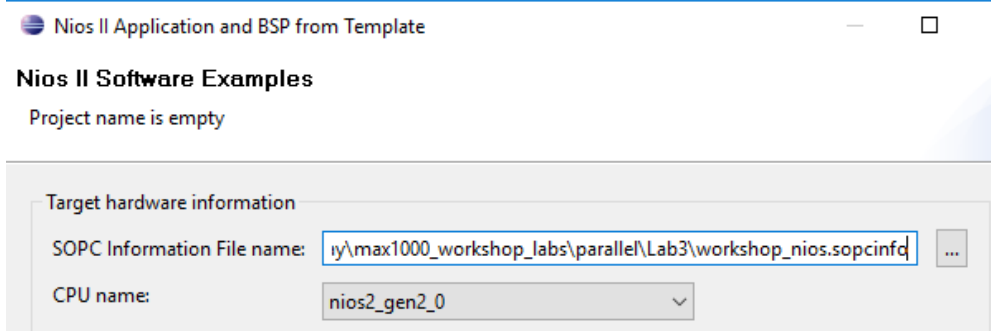
Our goal today is to build on a simple OS-less hello-world example and modify it to control our LED-counter.

3.3.1 Start NiosII EDS

- From the Windows Start menu or Quartus or Qsys Tools menu; start “Nios II Software Build Tools for Eclipse”
- When asked to select workspace, in this case we recommend inside the Lab3 project directory(but it can be anywhere)
- From the menu File => New select “Nios II Application and BSP from Template”

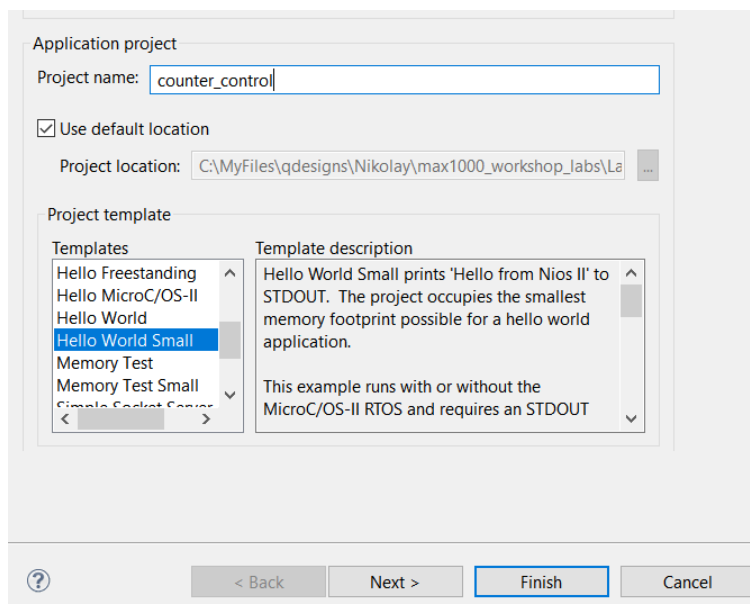


- Qsys created a .sopcinfo file that contains everything Eclipse needs to know about our custom processor system. Point to this in the SOPC Information File Name box.

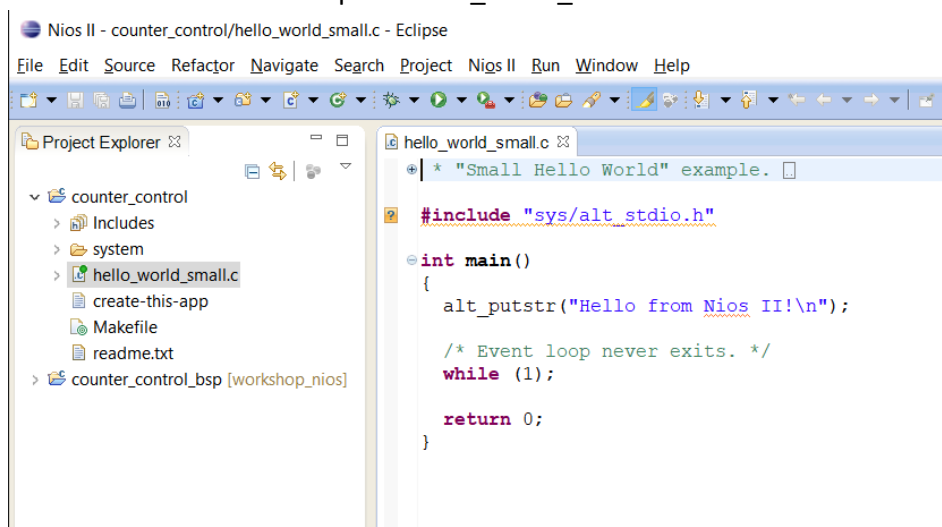


(Note: You may have several Nios II processors in a system, hence you select which CPU you are building software for)

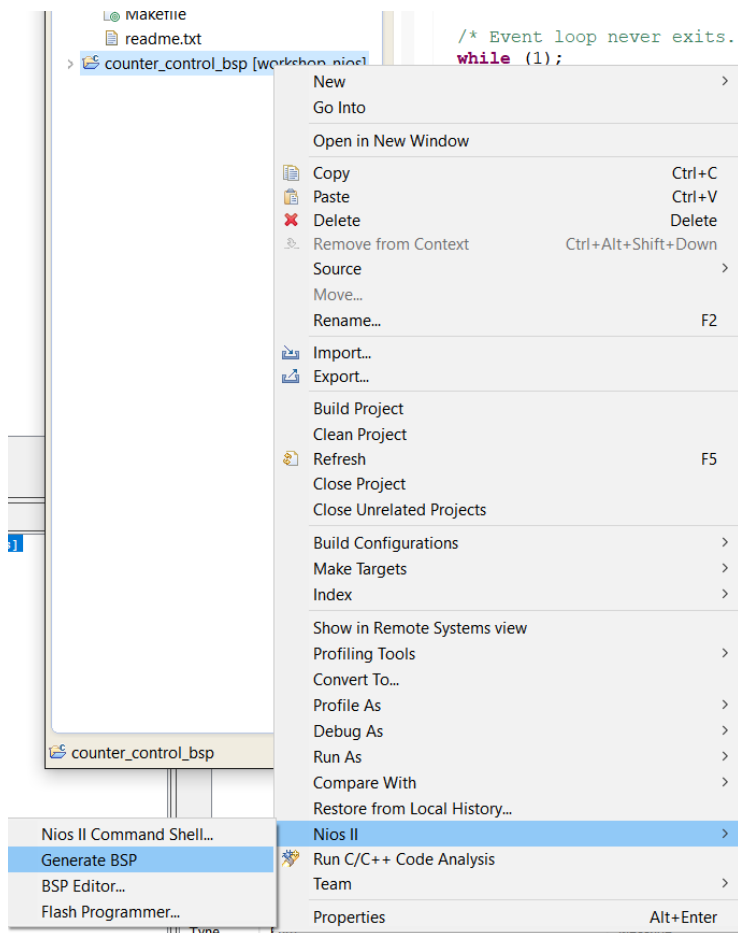
- Type in an Application project name of choice, and select the template “Hello World Small” since we want to keep the code inside the processors assigned 16kByte RAM in this case.



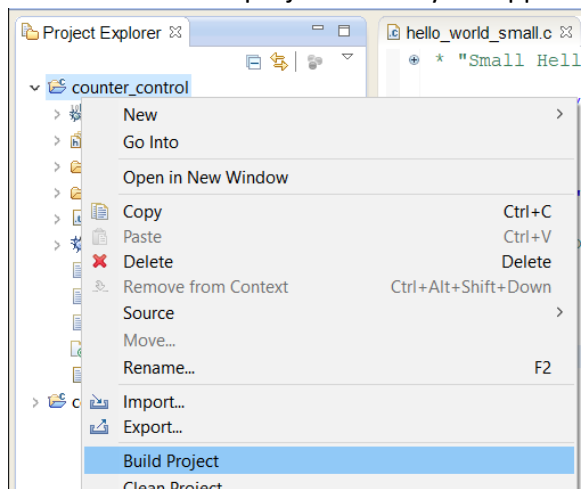
- You can now view the template `hello_world_small.c` code.



Note: If you make changes to your Qsys, you will need to update the BSP project to reflect the changes. This is done by right-click over the `_bsp` project and selecting Nios II => Generate BSP.

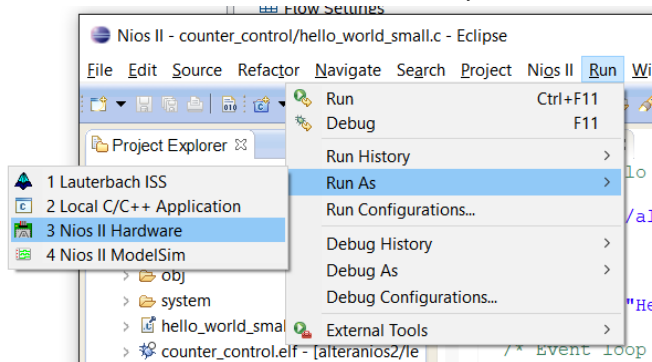


- Build the software project. Select your application project. Right-click and choose “Build”

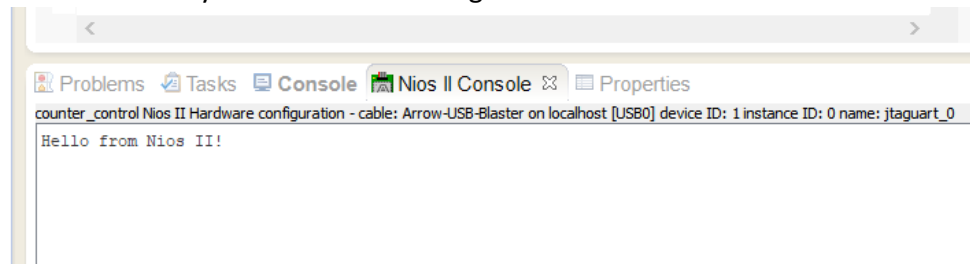


- Before running your software on the target, make sure that the fpga is actually configured with your desired processor system. The System ID peripheral contains a unique timestamp that is checked by Eclipse before attempting to run code.

- Now run the software on hardware by menu Run => Run As => 3 Nios II Hardware



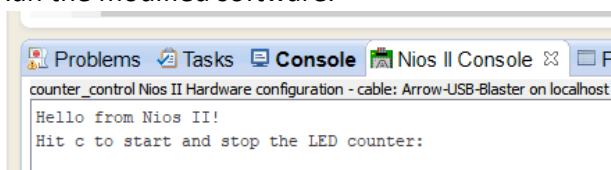
- With a little luck you will see something like this:



3.3.2 Modifying the software

Since we put in a hardware accelerator to control the led, we also want to control this from the software.

- Modify the hello_world_small to look like the code supplied in \Lab3\software\hello_world_small_control_counter.c
- Run the modified software.



- Click in the Nios II console window and type c + <enter>

You should now be able to turn the counting on and off.